NAG Toolbox

nag_sort_realmat_rank_rows (m01de)

1 Purpose

nag_sort_realmat_rank_rows (m01de) ranks the rows of a matrix of double numbers in ascending or descending order.

2 Syntax

```
[irank, ifail] = nag_sort_realmat_rank_rows(rm, m1, n1, order, 'm2', m2, 'n2', n2)
[irank, ifail] = m01de(rm, m1, n1, order, 'm2', m2, 'n2', n2)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: **m2** was made optional.

3 Description

nag_sort_realmat_rank_rows (m01de) ranks rows **m1** to **m2** of a matrix, using the data in columns **n1** to **n2** of those rows. The ordering is determined by first ranking the data in column **n1**, then ranking any tied rows according to the data in column **n1** + 1, and so on up to column **n2**.

nag_sort_realmat_rank_rows (m01de) uses a variant of list-merging, as described on pages 165–166 in Knuth (1973). The function takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal rows preserve their ordering in the input data.

4 References

Knuth D E (1973) *The Art of Computer Programming (Volume 3)* (2nd Edition) Addison–Wesley

5 Parameters

5.1 Compulsory Input Parameters

1: **rm**(ldm, **n2**) – REAL (KIND=nag_wp) array

ldm, the first dimension of the array, must satisfy the constraint $ldm \geq$ **m2**.

Columns **n1** to **n2** of rows **m1** to **m2** of **rm** must contain double data to be ranked.

2: **m1** – INTEGER

The index of the first row of **rm** to be ranked.

Constraint: **m1** > 0.

3: **n1** – INTEGER

The index of the first column of **rm** to be used.

Constraint: **n1** > 0.

4: **order** – CHARACTER(1)

If **order** = 'A', the rows will be ranked in ascending (i.e., nondecreasing) order.

If **order** = 'D', into descending order.

Constraint: **order** = 'A' or 'D'.

5.2 Optional Input Parameters

1: **m2** – INTEGER

Default: the first dimension of the array **rm**.

The index of the last row of **rm** to be ranked.

Constraint: **m2** $\geq$ **m1**.

2: **n2** – INTEGER

Default: the second dimension of the array **rm**.

The index of the last column of **rm** to be used.

Constraint: **n2** $\geq$ **n1**.

5.3 Output Parameters

1: **irank**(**m2**) – INTEGER array

Elements **m1** to **m2** of **irank** contain the ranks of the corresponding rows of **rm**. Note that the ranks are in the range **m1** to **m2**: thus, if the ith row of **rm** is the first in the rank order, **irank**(i) is set to **m1**.

2: **ifail** – INTEGER

ifail = 0 unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

On entry, **m2** $<$ 1,
or **n2** $<$ 1,
or **m1** $<$ 1,
or **m1** $>$ **m2**,
or **n1** $<$ 1,
or **n1** $>$ **n2**,
or $ldm <$ **m2**.

ifail = 2

On entry, **order** is not `A' or `D'.

ifail = -99

An unexpected error has been triggered by this routine. Please contact NAG.

ifail = -399

Your licence key may have expired or may not have been installed correctly.

ifail = -999

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

The average time taken by the function is approximately proportional to $n \times \log(n)$, where $n = \mathbf{m2} - \mathbf{m1} + 1$.

9 Example

This example reads a matrix of double numbers and ranks the rows in ascending order.

9.1 Program Text

```
function m01de_example

fprintf('m01de example results\n\n');

rm = [6, 5, 4;
      5, 2, 1;
      2, 4, 9;
      4, 9, 6;
      4, 9, 5;
      4, 1, 2;
      3, 4, 1;
      2, 4, 6;
      1, 6, 4;
      9, 3, 2;
      6, 2, 5;
      4, 9, 6];

m1 = nag_int(1);
n1 = nag_int(1);
order = 'Ascending';

[irank, ifail] = m01de( ...
                        rm, m1, n1, order);

fprintf('          Data             Ranks\n\n');
for i = 1:size(rm,1)
  fprintf('%7.1f',rm(i,:));
  fprintf('    %7d\n',irank(i));
end
```

9.2 Program Results

```
m01de example results

          Data             Ranks

    6.0    5.0    4.0         11
    5.0    2.0    1.0          9
    2.0    4.0    9.0          3
    4.0    9.0    6.0          7
    4.0    9.0    5.0          6
    4.0    1.0    2.0          5
    3.0    4.0    1.0          4
    2.0    4.0    6.0          2
    1.0    6.0    4.0          1
    9.0    3.0    2.0         12
    6.0    2.0    5.0         10
    4.0    9.0    6.0          8
```